OMOU Project Report

CPEN 291

Prof. Mieszko Lis

May 16, 2021

Max Cao, Ryan Clayton, Kaleb Hui, Sijan Poudel

Table of Contents

1. Introduction

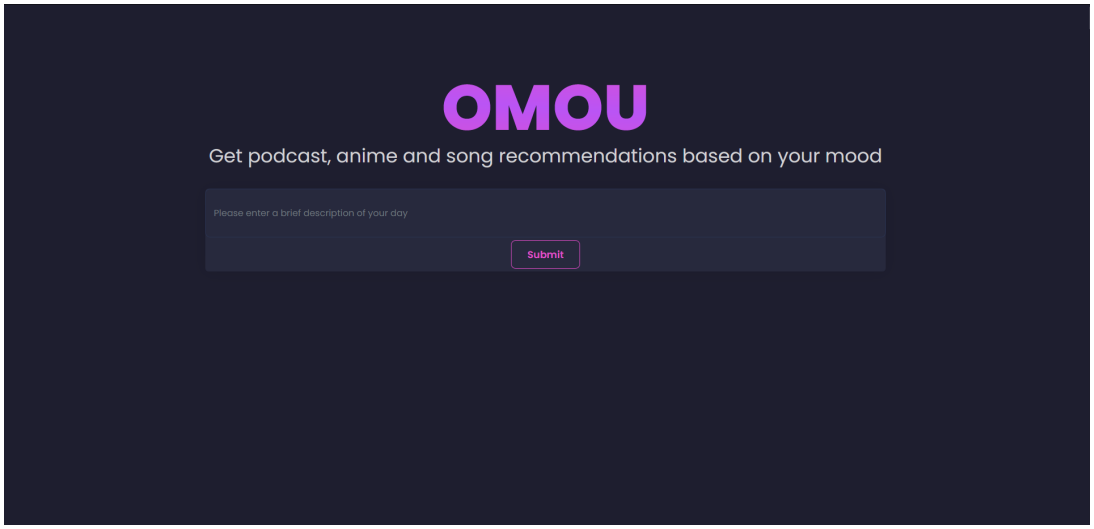
Studies have shown that the feeling of loneliness can be more lethal than smoking 15 cigarettes per day. The COVID-19 pandemic has had a huge impact on the majority of the population, leading people to feel lonely, isolated and depressed; these feelings can be hard to overcome [1]. For the CPEN 291 final project, our group has developed a web application to elevate the user's mood based on a personal journal entry that our product takes as an input. This application utilizes Machine Learning models to predict the mood from a given text input. The application will then be able to recommend anime, podcasts and music based on the user's current mood.

The objective of this report is to present the final outcome of our project. The report discusses different methods used to create the machine learning model, recommender systems, and the frameworks used to combine all of these components together. We begin the report by walking through the product itself, along with a description of how data flows through the components.

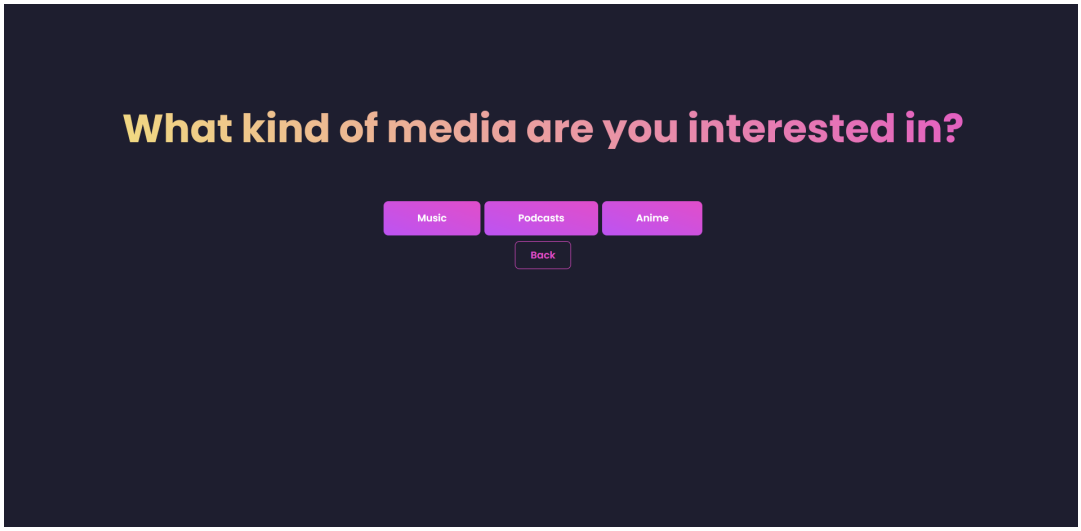
2. Project Description

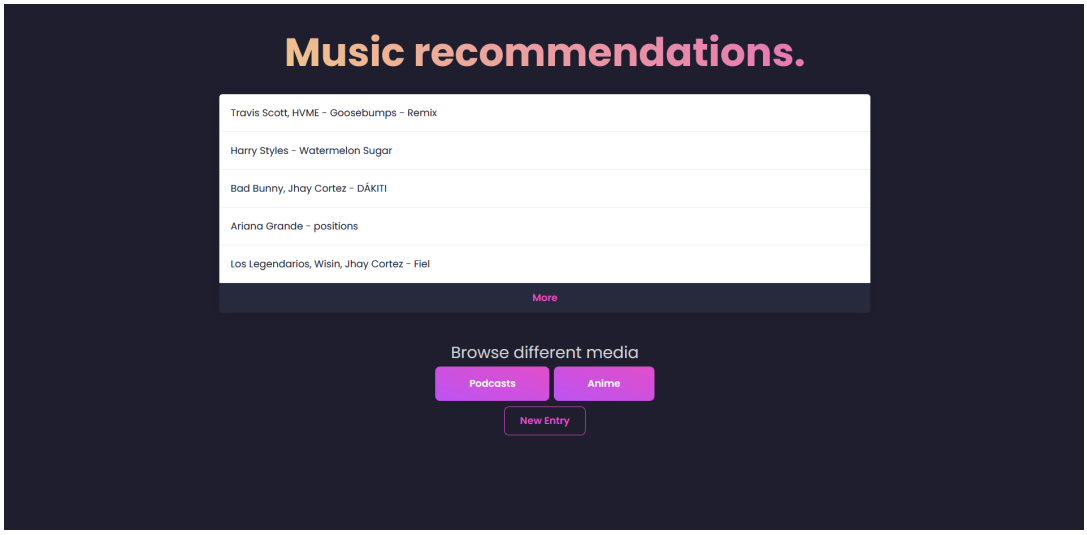
2.1 Product Walkthrough

Entering the website, the user is initially greeted by our front page which contains a prompt to enter a description of the user's day in a textbox.



Once a response is entered, the user input is passed to a function that cleans it, removing all punctuation, stop words and numbers. The cleaned input is saved as a session variable so that it is accessible in the other views. The website then moves to the next page which prompts the user to enter their media of choice. For our website, these options include Music, Podcasts and Anime. The user will select their media of choice and that will lead to another page.





Based on the user's selection the cleaned input will be passed to the function that implements the recommendation model for the chosen media. The selected function returns a list of all recommendations which are then displayed on the final page. This page will contain a list of the titles that accurately reflects a combination of their mood as well as their preferences. If the model returns a list containing more than five elements the user will have the option to cycle through them. The user will also be given the option to return to the front page and provide a new input or view different recommendations for other mediums using the current input.

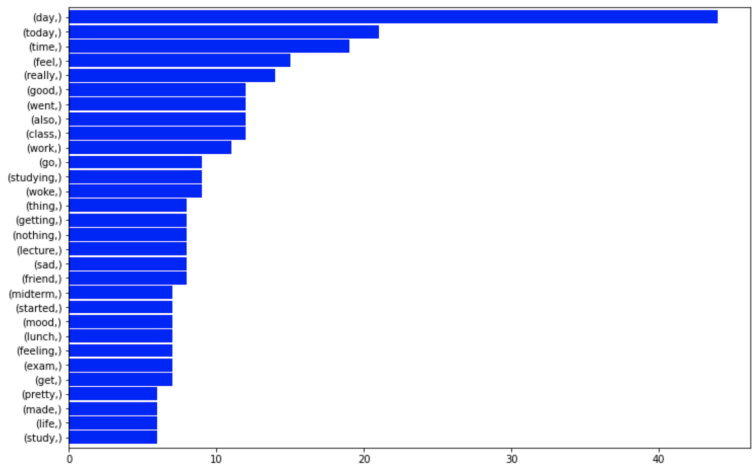
2.2 Component Diagram and Data Flow

The data flow of our product begins at the front page of the website. There, the user inputs a journal entry describing their day. This outputs a string that we clean by removing unneeded characters including punctuations, numbers etc. This string is then sent to the sentiment analysis model which outputs an emotion sentiment. Next, the user is given another prompt to provide their media of choice. With the emotion value and the chosen media, we then send the user value to the appropriate recommender system. The system then outputs recommendations based on the emotion value which is displayed on the website. All of these components are connected using the Django Framework.

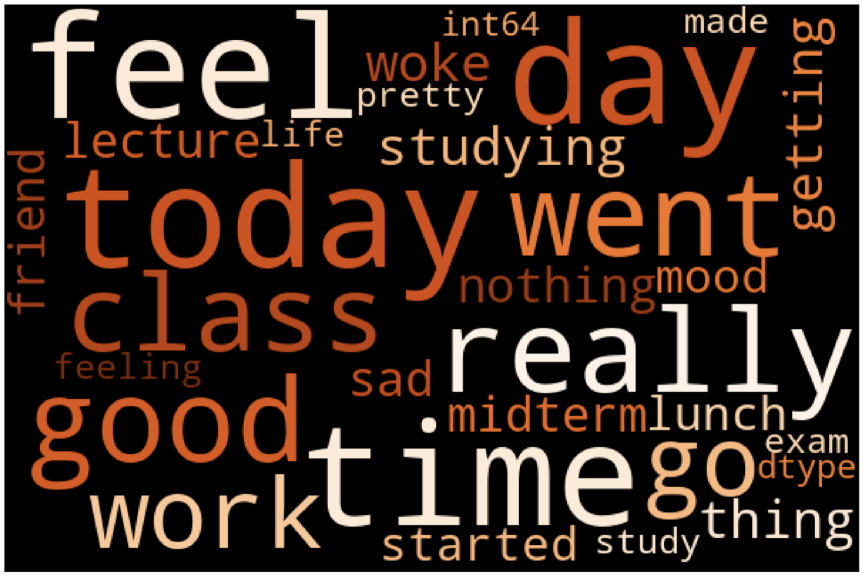
2.3 Real-Time Data Analysis

Initially, we conducted a survey using a Google form to collect a real-time dataset to study the feelings of people in the pandemic. We then did word cloud visualization using unigrams and bigrams, visualized the most common words in our dataset, and did topic modelling to gather information about our targeted users. Below are the results of our data analysis.

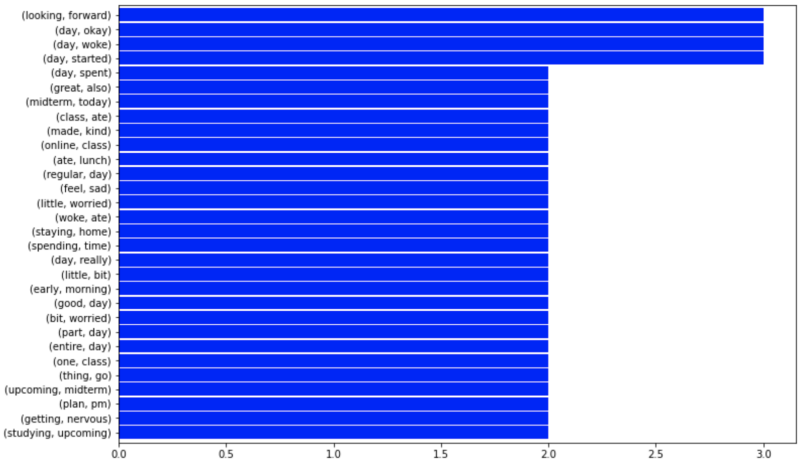
Unigrams:



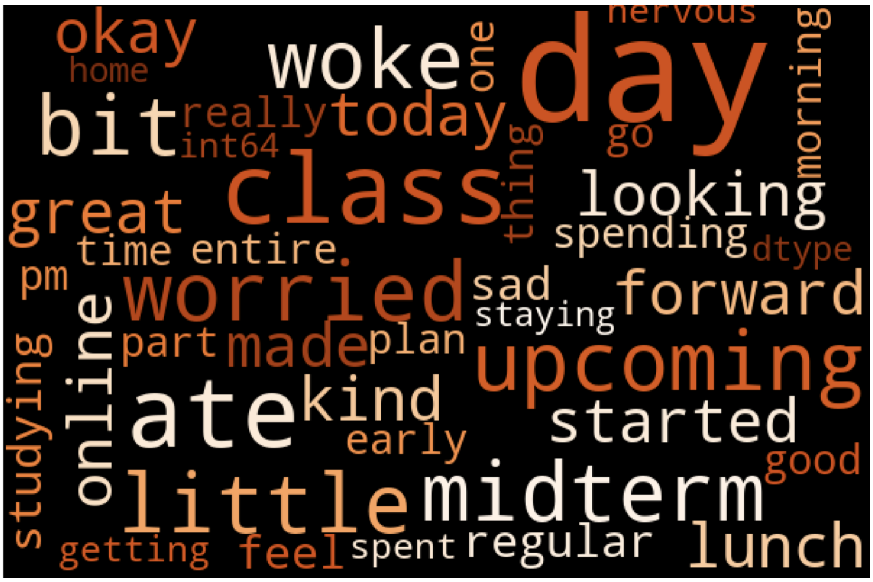
WordCloud Visualization (Unigrams):



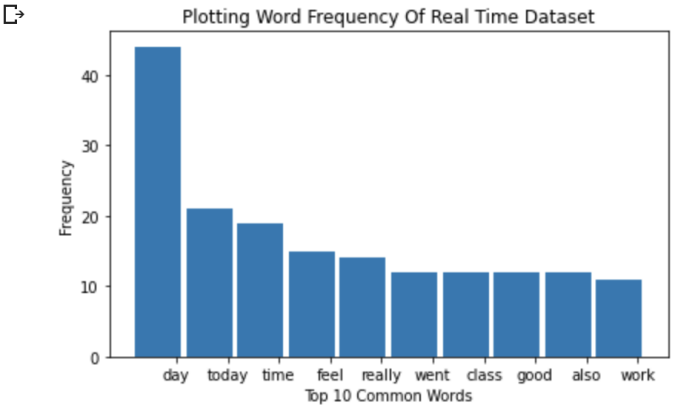
Bigrams:



WordCloud Visualization (Bigrams):

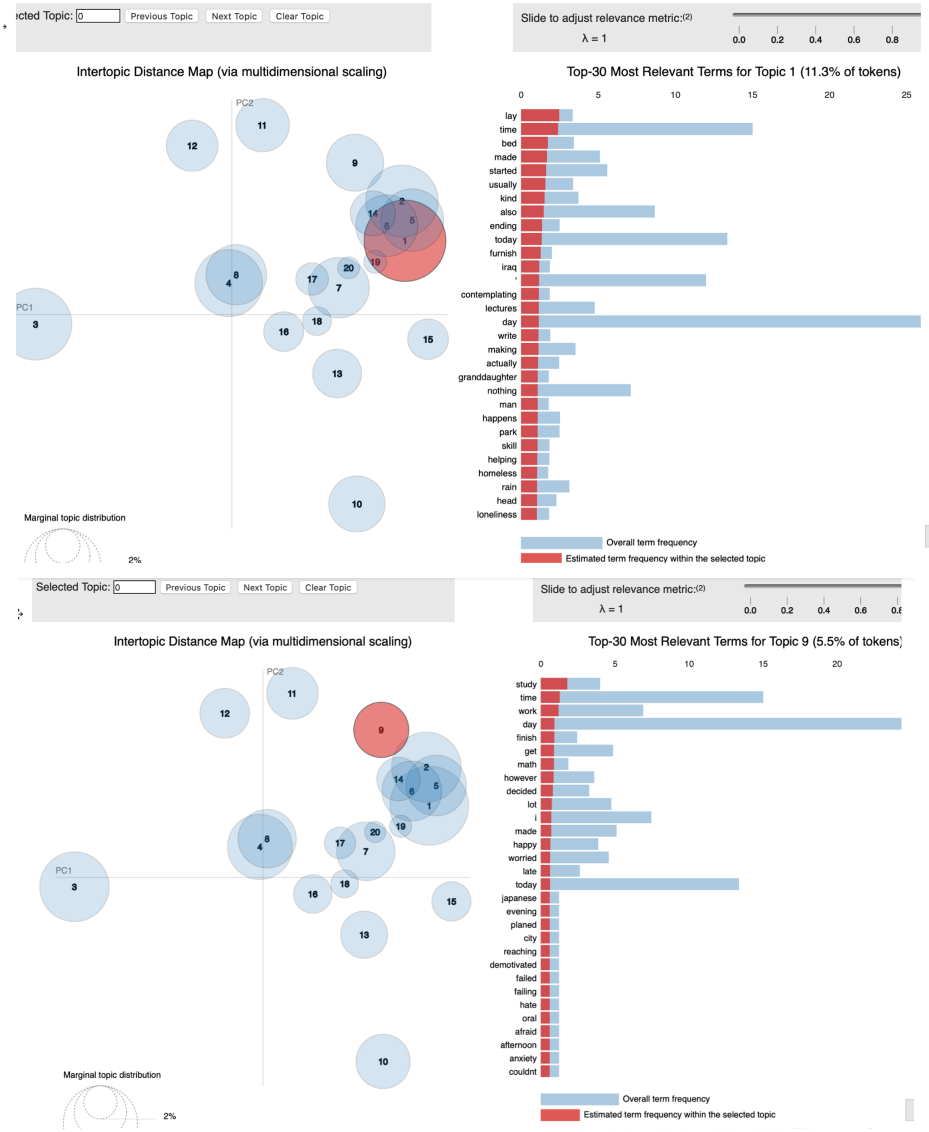


Plotting Common Words Frequency:



Topic Modelling:

The topic modelling was done using the Latent Dirichlet Allocation (LDA) technique. We used the gensim library to create a dictionary of corpus. We then analyzed the term frequency and topic word distribution of the input dataset. This was done by calculating the conditional probability of each and every word. The words with equal semantics and probabilities were then grouped together under one topic. In the end, about 20 topics were generated as shown in the pictures below.



Doing topic modelling, we visualized that most people were feeling lonely, depressed, anxious or worried. Visualizing the word clouds through unigrams and bigrams, we also realized many people were concerned about online studies/exams. We used this data as our motivation when moving ahead with the rest of the project.

3. Machine Learning Component

3.1 Sentiment Analysis Model

Our product required a Sentiment Analysis Model that could predict a user's mood based on a journal-like entry. Starting with the data used to train our model, we initially decided to create a survey and obtain data from other people. The survey would provide data in a format that we could easily use to train the model which was a journal entry along with the responder's mood. Although, we ran into obstacles with this method of data collection. Researching through sentence sentiment analysis models similar to ours, we discovered that we would require upwards of 10 000 entries to properly train the model. This was substantially higher than the number of responses we received from our survey. In consequence, we transitioned into a dataset named 'Emotions in Text' made by Ishant, a contributor on Kaggle [4]. After entries with non-essential categories, we had over 21 000 entries and 6 different mood labels including "sadness", "anger", "fear", "love", "surprise" and "happy".

The model was built with PyTorch libraries and utilized Natural Language Processing (NLP) techniques. Originally, we planned to create this model using Naïve Bayes-based Support Vector Machine (SVM) for sentiment analysis. Although according to models used for tasks similar to ours, they generally performed better using LSTM based NLP [2, 3]. Since our model used NLP we needed to first convert the input sentence to a form that it could properly interpret and analyze. To do this, we tokenized each sentence before it was inputted into the model using the tokenizer from the SpaCy library. The model itself consisted of three main layers. The first utilized the word embeddings, next we used the LSTM architecture as a layer and lastly a linear layer. The training used cross-entropy loss as that helped the model detect how close or far its predictions were.

Finally, the model was trained on the dataset mentioned above. By tuning various hyperparameters, adjusting the different dimensions for each layer in the model and trying out different optimizers and schedules we were able to achieve an accuracy of 91% in 20 epochs. Admittedly the accuracy we achieved could be further improved, although due to time limitations we decided to work on other parts of the project. The complete list of hyperparameters used in our final model is listed below.

```
model = SentenceModel(len(ds_full.vocab), 64, 64, len(text.Emotion.unique()))
device = torch.device('cuda:0')
model.to(device);
crit = nn.CrossEntropyLoss().to(device)
opt = optim.SGD(model.parameters(), lr=0.1)
sched = optim.lr_scheduler.StepLR(opt, 1, gamma=1)
```

4. Recommender Systems

Using Python, we developed separate recommender systems for three different types of media, providing more flexibility and choices for the different users. We chose to recommend music, podcasts, and anime. We felt that three was enough for the user, although we would've liked to have an extra system for movies, however, due to time constraints, scheduling and the overlap between movies and anime we decided it was best to not roll out the extra feature and allocate time elsewhere.

4.1 Music

For the recommendation of music, we were looking for a large dataset of music that had metrics related to the music itself. We found a very good dataset curated by a user Yamac Eren Ay on Kaggle, which uses the Spotify API to gather tracks into a dataset containing about 600,000 songs [5]. The Python data analysis library Pandas was used to transform our dataset .csv file into a workable data structure, (called a data frame), in our code. The dataset contained various metrics that were quantitatively defined by the Spotify API, (e.g. danceability, valence, acousticness etc.) We decided that some of these metrics were not useful to our analysis (such as loudness, tempo, time signature) and removed those. We also decided to cut down on the number of tracks in our dataset to around 22,000 by cutting out tracks that were considered to have less than "65 popularity", which was also defined by the dataset.

Our system first uses our user's input and converts it into a 6-index word embedding tensor using our SentenceModel Dataset. The embedded tensor is then passed into our model which creates a new tensor with various weights for the 6 sentiments. Our model usually predicts the sentiment by using the argmax function to use the highest weighted index of the tensor as the output. Hence, these weights should be some indication of how much of each emotion is in the input. So we used ReLU and the logarithm function on the weighted tensor to "normalize" each weight. Finally, the Softmax function was used in order to make sure the sum of the weights add to 1 for more normalization. This final tensor is then used as a distribution of sentiment in each input. We then developed an algorithm to generate "scores" for all tracks in our sub-dataset. The scoring algorithm is described at the bottom of the Music subheading.

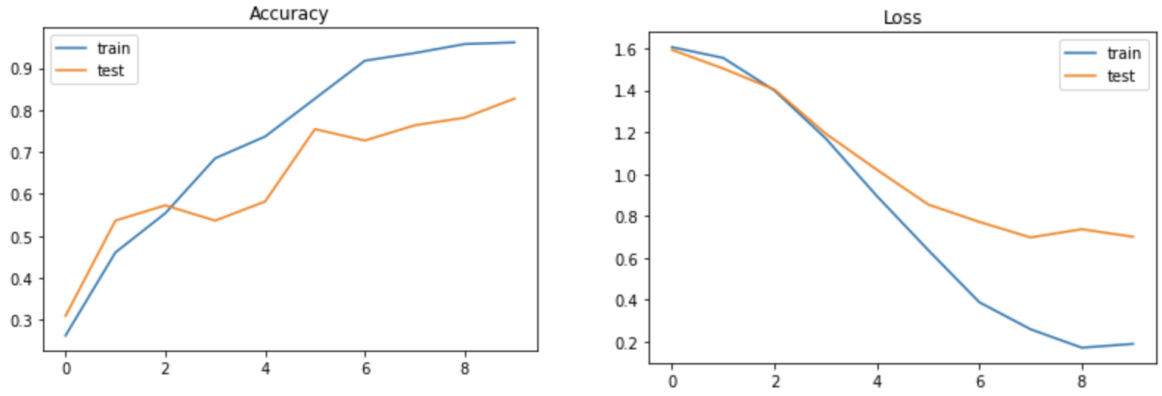
These scores were then used to sort the dataset from highest to lowest and the top 25 highest scoring tracks were filtered into a new Pandas "Series" data structure (a row in a Pandas dataframe). Then finally, NumPy's random.choice function is used to select 5 random music tracks from the top 25 tracks.

TRACK_SCORE =
POPULARITY_WEIGHT * (track.popularity / 100) +
in.sadness * (1 - track.valence) +
in.anger * track.energy +
in.love * (mean(track.energy + track.danceability)) +
in.surprise * (1 - track.energy) +
in.fear * valence +
in.happy * (mean(track.danceability + track.energy))

1. "POPULARITY_WEIGHT" is a programmer defined constant that can be changed based upon how more important we feel that more "popular" songs are recommended compared to less "popular" songs
2. "track" is a single music track in our sub-dataset
3. "in" is the input distribution sentiment tensor
4. "mean" is a function which takes the average mean of two or more numbers

4.2 Podcast

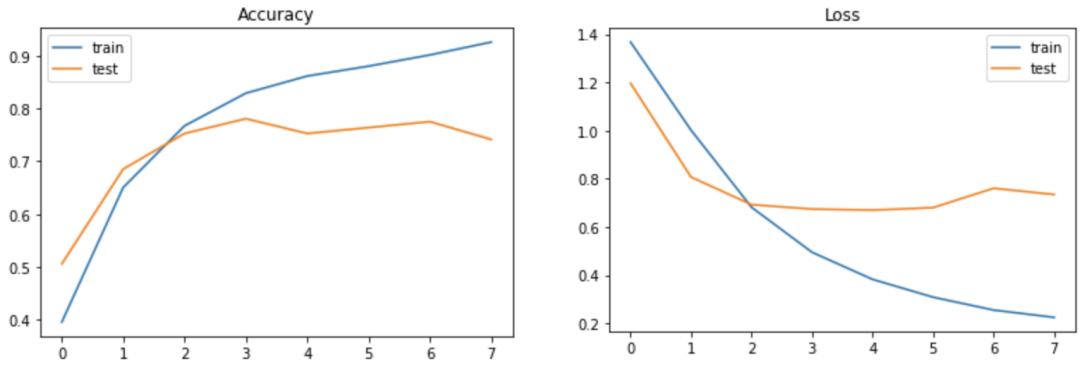
For the recommendation of a podcast, we used the iTunes podcast dataset created by Roman6335 [6]. The dataset consists of a column with descriptions of over 13000 podcasts. These fall under 68 different genres with different quantities of podcasts in each genre. The LSTM model was then applied to the dataset. For the multilevel classification, as the number of classes increases the accuracy decreases. As a result of this, classifying all of the 68 genres with optimal accuracy was challenging. So we decided to pick the genres that were most relevant to our targeted user: those that had more value counts and that gave better accuracy after training. The five genres we decided upon were ‘Literature’, ‘Business News’, ‘Comedy’, ‘History’ and ‘Places and Travel’. After merging our selected genres, the description was cleaned to remove special characters, symbols and stopwords. We then tokenized our dataset and passed the tokens to our model. The model consists of several layers with the first layer being the embedded layer which consists of a 100 length vector to represent each word. The next layers are the LSTM layer followed by the output layer which creates five output values, one for each genre. We use softmax for the activation function, Adam for optimizer and categorical cross-entropy for the loss function. Using a batch size of 64 within 10 epochs our podcast recommendation model was able to achieve an accuracy of 82%. Below is the plot showing improvement of our training accuracy and test accuracy as well as training loss and test loss for podcast recommendation as a function of epochs.



Finally, our model predicts one of the five genres and gives a series of 20 different podcasts which fall under the predicted genre. We then select five of them to give recommendations to the user.

4.3 Anime

For the recommendation of anime, we used Anime Movies, OVA's and Tv-shows dataset created by Thomas Konstantin [7]. This dataset consists of a column containing synopsis of over 16000 anime and tv shows. We then used the same process as the podcast dataset for our recommendation. Again, there were over 4800 different genres for the anime dataset. We narrowed down our dataset to only have genres that had the most contained the most titles and were the most relevant for mood elevation. Thus, we picked animes which fell under the genre of 'Music', 'Comedy', 'Fantasy', 'Drama' and 'Dementia' as our training dataset for our model. We used the LSTM model and got the best accuracy using Adam optimizer, categorical cross-entropy for the loss function and SpatialDropout1D to perform variational dropout in our model. Using a batch size of 32, within 15 epochs our anime recommendation model was able to achieve an accuracy of 78%. Below is the plot showing improvement of our training accuracy and test accuracy as well as training loss and test loss for anime recommendation as a function of a few epochs.



Similar to the podcast recommendation, our model then predicts one of five selected genres in the anime dataset and returns a series of 20 different anime which fall under the predicted genre. We then recommend 5 of them to the user.

5. Non-Machine Learning Component

5.1 Web Application

Originally, we planned on developing a mobile app using Kivy, an open-source Python framework. However, after beginning to work with Kivy it became apparent that it was no longer well supported and that the limitations of the libraries would make development more challenging. We decided it would be easier to produce a web application using Django. Our decision to use both Kivy and Django was heavily influenced by them being python frameworks. As none of us had any experience using any of the alternatives we didn't think it would be practical to try learning an entirely new language in the given time. The frontend component for each of the views was done using HTML and the templating engine Jinja. Jinja allows dynamic values to be used in HTML, and, when combined with Django's built-in template tags and filters, it allows more complex code such as conditionals and loops to be added to HTML. The CSS file used was taken from Black Dashboard, a free Django template by creative-tim [8]. Some small modifications were made to the CSS file in order to change the positioning of the elements and two new elements were added to change the style of headers.

6. Contributions

Kaleb: Designed and trained the NLP Model (section 3) used to perform sentiment analysis on the user's input. Assisted with music recommendation by creating a dataset used to map the NLP model's output to a song though this idea was scrapped later on in the project for a more sophisticated system (section 4.1).

Max: Used the trained NLP Model and the Kaggle Spotify dataset to create the music recommendation system and its algorithms (section 4.1). Also assisted with training the model itself (section 3) as well as with data preprocessing (section 2.3).

Ryan: Built the web application (section 5) to receive the user's input, facilitate data flow and display the outputs returned from the other components.

Sijan: Did data preprocessing and analysis in the real-time dataset and the emotion dataset (section 2.3). Created and trained the model for the podcast (section 4.2) and anime (section 4.3) recommendations for users based on their input.

7. Conclusion

The goal of this project was to design a product that utilizes machine learning (ML) to analyze a non-ML input and produce a non-ML output. For many, loneliness often leads to a low mood or feelings of depression. During the current pandemic, studies indicate that there has been a rise in social isolation, affecting people around the globe [1]. With this real-world problem and project goal in mind, we decided to help alleviate this problem by developing a product that can recommend content for a desired medium. This resulted in the creation of a fully functional web application that can accurately provide recommendations to the user matching the input journal entry and their media preferences.

8. References

8.1 Citation

[1] C. Walsh, “Young adults hardest hit by loneliness during pandemic” *The Harvard Gazette*, Feb 21, 2021.

[2] G. Loye, “Long Short-Term Memory: From Zero to Hero with PyTorch” *FloydHub*, Jun 15, 2019.

[3] Aakanksha NS, “Multiclass Text Classification using LSTM in PyTorch”, *towardsdatascience*, Apr 7, 2020.

8.2 Resources Used

[4] Emotions in Text Dataset by Ishantjuyal:
https://www.kaggle.com/ishantjuyal/emotions-in-text

[5] Spotify Kaggle dataset by Yamaerenay:
https://www.kaggle.com/yamaerenay/spotify-dataset-19212020-160k-tracks/activity

[6] iTunes Podcast Dataset by Roman6335:
https://www.kaggle.com/roman6335/13000-itunes-podcasts-april-2018

[7] Anime Movies, OVA’s and Tv-shows Dataset by Thomas Konstantin:
https://www.kaggle.com/thomaskonstantin/top-10000-anime-movies-ovas-and-tvshows

[8] Black Dashboard by creative-tim:
https://demos.creative-tim.com/black-dashboard/